EXHIBIT R

EOS CLI Conventions and Style Guidelines

Introduction

The Arista EOS command line interface closely follows the industry standard command line interface in its syntax and behavior. This makes it easier for our customers to use an EOS device with little or no training, since most of them are already familiar with the industry standard. In many cases, the functionality we are building is already present in the industry standard, and our customers benefit when we use the same configuration and show commands. As we add more functionality to EOS, though, we have to create our own commands for configuring and managing these new features. For those, it is important that we keep the new commands consistent in style and behavior with the rest of the CLI. This consistency makes it easier for customers to use our devices and provides a more pleasant experience in doing so.

This document specifies many of the conventions we use in adding commands to our CLI. These are guidelines, not laws, though, so we still have to think as we add new commands. One rule we try hard to follow is to review new CLI commands on the cli-review@ mailing list. When you are coming up with a new set of commands for managing a new feature, please send your proposal to cli-review@. Please do this early in the process, as our experience with this is that many proposals change significantly during the discussion. See the section on sending requests to cli-review@ for some tips on how to do this successfully.

Some Philosophy

Our style of configuration CLI has a bit of a philosophy behind it. Configuration commands are attribute-oriented. The commands set attributes to values or unset them back to a default value. Much like with our attribute-oriented programming style, this means that configuration commands are named as nouns, not verbs. Just like in attribute-oriented programming, we are specifying the configuration state with our commands, not specifying how or when the switch should implement the behavior needed to be consistent with that configuration state.

The configuration CLI is also object-oriented. In many cases, we use a configuration sub-mode to create an object and configure attributes of that object. Interface mode is the most obvious example. We configure the attributes of an interface by entering interface mode using the interface name as a key and then applying commands that apply to that interface object. The alternative is to use global commands that name the object whose attribute is being assigned to a new value. There are examples in the CLI that do this, the IGMP Snooping feature is a prominent example, but we are trying to avoid that pattern and follow the object-oriented, mode-

oriented model instead.

Conventions and Guidelines

Don't Change Existing Commands

Once a command is shipped to customers, whether a config command or a show command, we should be very reluctant to change it. For config commands, changes introduce painful compatibility issues. We have to worry about compatibility with startup config files generated by an old release and being parsed by the new release. We also have to worry about startup config files generated by the new release and being parsed by an old release after a downgrade. This is a real pain to manage, and it is almost never the case that the right answer is to change the existing configuration commands.

We used to have more flexibility with show commands. Now, though, we use our show commands as operations for the HTTP-JSON Commands EAPI, so the commands are getting hard coded into programs used to monitor our switches. Breaking those programs must be a last resort.

If you think you should change the syntax of an existing command, please try to come up with some way not to do so. Then raise the issue on cli-review@, and be very explicit about the fact that you are considering changing the syntax of an existing command. These changes must get serious consideration from multiple people before going through.

Follow the Industry Standard

The first and probably most important convention in our CLI is to follow the industry standard. If the commands for a given feature are already out there in the industry, we don't add value by coming up with another command model that is similar but different. All we accomplish in doing that is to make it harder for our customers to learn how to use our switches and routers.

In looking for industry standard models to follow, please look in the following order of preference: IOS, NX-OS, IOS-XR, JunOS.

If the industry standard CLI model is truly terrible, we can look at deviating from it. This doesn't happen very often, but like most of our rules at Arista, even this one is open to using our judgment.

Avoid "enable" and "disable" keywords

The "enable" and "disable" keywords are strongly frowned upon in industry standard style CLI. It is true that some of them have crept in over the years, but that doesn't mean we should add more. Try hard to find ways to express your configuration in a way that doesn't require an on/off switch. When one is required, the "[no] shutdown" pattern is generally the right answer.

I think this really originated with the 'enable' keyword being redundant. You had 'foo' and 'no foo' in the CLI, so what value does 'foo enable' add? The pattern we try to follow is that you assert that something is enabled by specifying its existence in the configuration, and you disable it by removing it from the configuration.

These cases where there are attributes of the feature to be set separately from enabling the feature seem to often put us in this case where the 'enable' keyword is tempting. We have '[no] foo' and also '[no] foo attribute <value>'. The commands really configure separate attributes in the configuration, one being the enabling of the feature and the other being some parameter used by the feature. While it can be confusing when the configuration shows 'foo attribute val1' and the feature is still disabled, I don't think adding 'enable' keywords really eliminates the potential for confusion. Usually, what you need here is a better show command that clearly states that the feature is disabled. Beyond that, we have to rely on documentation and eventually our helpful customer support folks :-).

Example

Don't copy

spanning-tree bpduguard [enable]

Instead try

spanning-tree bpduguard

The extra "enable" keywords really didn't add anything.

Avoid-excessive-dashes

It is often tempting to combine multiple words together in a CLI command using dashes, "-". While combining two words that truly form a single concept can be OK, if you find yourself combining more than two words it is likely time to think more about what you are doing. There are a couple of problems with multi-word dash separated tokens in the CLI:

- They break the CLI's ability to understand token prefixes. If there are two tokens "foo-bar" and "foo-baz" that are valid at the same level, the CLI cannot distinguish them until the first differentiating character is typed. This reduces the user's ability to shortcut tokens in the cli by typing just a unique prefix for the token and letting the parser figure out the rest.
- These just get awkward. Objects and concepts in the CLI need good names, and multi-word concoctions usually indicate that we haven't worked hard enough to come up with a good name for the thing being configured.

Example

Don't copy

ipv6 address use-link-local-only

Instead try

ipv6 address link-local

Command tokens should not be complete sentences :-).

Consistency is Good

Look for names and patterns in the existing CLI commands that match the problem you are trying to solve. By using consistent names for the same things in different features, we make it easy to understand what a command configures. By following the same patterns and structures, we make it easy to figure out how something should be configured.

Example

Don't copy

monitor link-flap
enable

Instead try

monitor link-flap
shutdown

I'm not trying to beat a dead horse here, but using "shutdown" for an explicit on/off switch, as awkward as it may seem at first, keeps our features consistent across a broad spectrum of features going back many years.

"no" and "default"

Typical configuration commands can be reverted to their default values by prefixing them with either the "no" or the "default" prefix. In most cases, be sure to support both of these for your new commands. There is a whole AID about this, AID 986, so I won't go into detail here about this topic.

Avoid New Acronyms

We try to avoid introducing new acronyms in the CLI, because only we know what they mean. Industry standard acronyms, for example "ip" and "qos", are OK, but try to avoid new ones. If we name a new feature by an acronym and document it well enough to make it a new, well known term, then we can use that, too. Be careful in doing that, though, as it can be easy to think that something is obvious when it is not.

Example

Say that you're adding a new feature "Fast Server Failure Detection", which you refer to as "FSF" internally. Instead of configuring it with:

fsf

Instead try something more spelled out:

monitor server-failure

Use Modes and Hierarchy

We are trying to avoid putting more and more commands at the global config level. When adding new commands, consider whether you are creating new configuration objects and

setting attributes on them. If so, look at creating a new configuration sub-mode for your commands. Even if the only object involved is the singleton configuration object of your feature, try creating a singleton mode (no "key" provided on entering it) to add your new commands within.

Multiple levels of modes are OK, too. Our support for these is improving, and they help to identify the objects being configured in the naturally nested structure of many configuration models.

Example

Don't copy

ip igmp snooping vlan 10

Instead try

vlan 10

ip igmp snooping

Keep Configuration Commands Order Independent

People are often tempted to protect the user by placing ordering constraints among commands in the CLI. The thinking is that if we make it so that you can't enter command X until command Y is present, we'll prevent the user from misconfiguring the switch since command X only makes sense when command Y is present. While that's true, we also introduce an ordering constraint into the startup config when we do this. One too many of these, and we have a constraint loop that we're really not going to be happy with.

Instead, we allow users to enter commands in any order. If a given command doesn't make sense yet, we just ignore the state that it configures until something else is configured and it does make sense. We don't print a warning that this is the case either, as these get obnoxious far too easily. It is important to have good show commands displaying the current operational state of each feature, so that when something is partially configured and that configuration is inactive the user can see what is actually going on.

Avoid Printing Warnings

As described above, people are often tempted to print a warning to the command line whenever the user enters a command that seems to not make sense. In general, we try to avoid doing this. These warnings might be helpful, but they can also become irritating if the user knows what they are doing. We err towards assuming that the user knows what they are doing. Our users deploy partial configurations for many reasons, so don't assume that you know better than our users and tell them a configuration is incorrect. Instead, add output into your show commands to indicate whether the current config is fully active and consistent.

Keep Commands Independent

Just like in attribute-oriented programming, it is best if each command controls the value of a single attribute or set of attributes in the configuration and no effect on the attributes controlled by other commands. I think of this as orthogonality among commands. When a command for one configuration attribute goes and changes some other attribute, the results are usually unexpected. Surprising people in the CLI is a bad idea. This is sometimes put forward as a convenience, as in "The 'no foo' command will delete all of the foo related parameters." This is not how we like to structure things. It reduces the flexibility of the CLI in the name of convenience, which we believe is the wrong tradeoff.

Example

Today in EOS

interface eth1

switchport access vlan 5

creates vlan 5 if it is not already present. It just shouldn't do this. The switchport command should not be reaching around and creating vlans controlled by the vlan command.

Use Conventional Command Hierarchies

When naming a new command submode, try to see if you can follow an existing pattern and use an existing naming convention. For example, we now have several flavors of **management** config modes: **management ssh, management telnet, management defaults.** Placing the configuration of features related to the management of the switch under config modes all starting with the **management** keyword makes the similar nature of these functions obvious in the CLI.

Some other common config mode prefixes are: **interface, vlan, router, and monitor.**

Show Command Suggestions

1. Make your command work with CAPI.
2. It is often helpful to provide show commands that display the "active configuration" of your feature in addition to its runtime status. Configuration can be inactive for a number of reasons, and making it clear in a show command that the configuration is having no effect can be very useful. This is more effective than the sometimes suggested mechanisms of log messages and warnings issued as configuration is entered.
3. Try to limit the text output of your commands to 79 columns, at least in the normal case.
4. Prefer "Interface" to "Port" for labels.
5. When creating columnated show screens underline the column header.
6. Use long names 'Ethernet1/2/3' vs short name 'Et1/2/3' when space allows.

Example

```
fm225# show int et1 mac
Key:
```

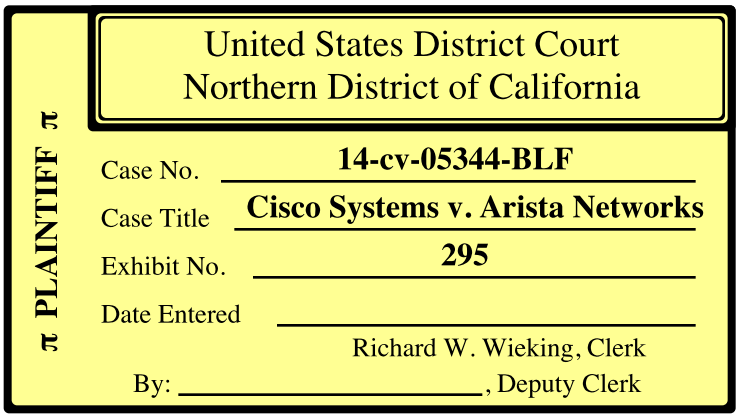
```
   L  = Rx Local Fault Reported
   R  = Rx Remote Fault Reported
   Last Change: Time when most recent fault status changed

                    Config Oper                        MAC                Last
Interface           State  State    PHY State          Fault            Change
------------------- ------ -------- ------------------ ----- -----------------
Ethernet1           Up     linkUp   linkUp                       4:49:02 ago
```

Tips for Sending Email to cli-review@

When you send your proposed new commands or command changes to cli-review@, please keep the readers of your email in mind. Also, the cli-review@ list is not for code reviews. It is for command syntax reviews. Please keep that in mind as you write your email.

- Clearly state whether the commands are industry standard or not. Don't make us look up your commands to figure this out.
- Clearly specify the new syntax. Don't hide it in a mess of CliPlugin tokens or broken down token by token with the help messages of each displayed.
- Don't send us a link to your review-board review. We don't want to go find your commands among a thousand lines of diffs.
- Don't send us only a link to your design document. A link can be handy, but please copy the new commands you need reviewed directly into the email you are sending.
- When asking about show commands, please provide sample command output directly in the email you are sending.
- Silence is not approval. If you don't get a response, please send an email asking for a response. If all else fails, poke asweeney@ or hzhong@ directly.
- If a conversation falls quiet, that does not necessarily mean that it has converged. People get busy. Make sure that you get a positive confirmation that your review is complete.


PLAINTIFF
United States District Court
Northern District of California
Case No. 14-cv-05344-BLF
Case Title Cisco Systems v. Arista Networks
Exhibit No. 295
Date Entered
Richard W. Wieking, Clerk
By: , Deputy Clerk